A GRAPH COLORING ALGORITHM AND A SCHEDULING PROBLEM

Robert Albert Draper

United States
Naval Postgraduate School

THESIS

A GRAPH COLORING ALGORITHM AND A
SCHEDULING PROBLEM

by

Robert Albert Draper

Thesis Advisor: U.R. Kodres

June 1971

A Graph Coloring Algorithm and A Scheduling Problem

by

Robert Albert Draper
Lieutenant, United States Navy
B.S., Northern Illinois University, 1963

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1971

ABSTRACT

The graph coloring problem is defined, and its importance in several applications is noted. A new algorithm to color graphs is presented and tested against the Welch-Powell algorithm. Significantly better results are obtained on a sequence of three hundred randomly generated graphs.

The new algorithm is applied to the solution of a scheduling problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. THE GRAPH COLORING PROBLEM

An intuitive concept of a graph is that of a set of points which are related to each other by connections or curves between some pairs of the points. As examples: a system of highways can be thought to be a graph. Highway junctions are the points, and the connections between points are the roads between the junctions. A computer program flowchart is a graph in which the points are the problem steps and the connections are the paths of program flow from step to step. A political map is a graph if the states are the points, and the connections are the common borders between neighboring states. See Figures 1 and 2 for pictorial demonstration.

An important problem in graph theory is the graph coloring problem. This problem is to color each points of an arbitrary graph using as few colors as possible and subject to the constraints that all the points are colored with one and only one color, and such that for any two points which have the same color, there is no connection between them. It is this coloring problem which is investigated in the thesis.

B. THE USES OF GRAPH COLORING

Political maps are usually published so that adjoining countries are colored differently. Such a coloring is equivalent to coloring the points of the graph associated with the map. Figure 2 is a map along with the associated graph and a coloring.

An interesting conjecture is that only four colors are needed to color all the countries on any map. This is a famous unsolved problem which was first stated in 1850.

A practical problem is to schedule the final examinations for a university such that no student will be assigned two finals during the same period, and such that the total time allocated for all the examinations is a minimum. The solution: consider each class examination to be a point of a graph. Let two examinations be connected if there is at least one student who will take both examinations; that is, there is conflict between the two classes. Then assign colors to the class examinations so that if two classes conflict they are assigned different colors, and such that the number of colors used is a minimum. Translate the colors into time periods and the schedule is written.

A further example consists of airport control towers, which require radio frequencies in order to operate. Consider the graph of airports. Two airports will be connected if there is insufficient distance between them to ensure that there can be no mutual radio interference between them. Then the assignment of radio frequencies to airports can be considered as a graph coloring problem.

C. OUTLINE OF THE THESIS

Section I of this thesis presents the intuitive meaning of the words "Graph" and "Graph Coloring," and discusses the importance of the problem.

Section II deals with the fundamental concepts of graphs - precise definitions of graphs and some properties of them. The problem of coloring a graph is specified with more precision and certain basic definitions related to this problem are stated. One of the methods for obtaining a coloring, namely the Welch-Powell [Ref.1], is discussed.

Section III describes an algorithm which is proposed as an improvement over presently known methods. On the average this algorithm will produce colorings of fewer colors than the Welch-Powell method described in Section II.

Section IV is a summary of tests comparing this new algorithm to the Welch-Powell algorithm.

Section V details a class scheduling problem and offers a means of solving it with the use of this proposed algorithm.

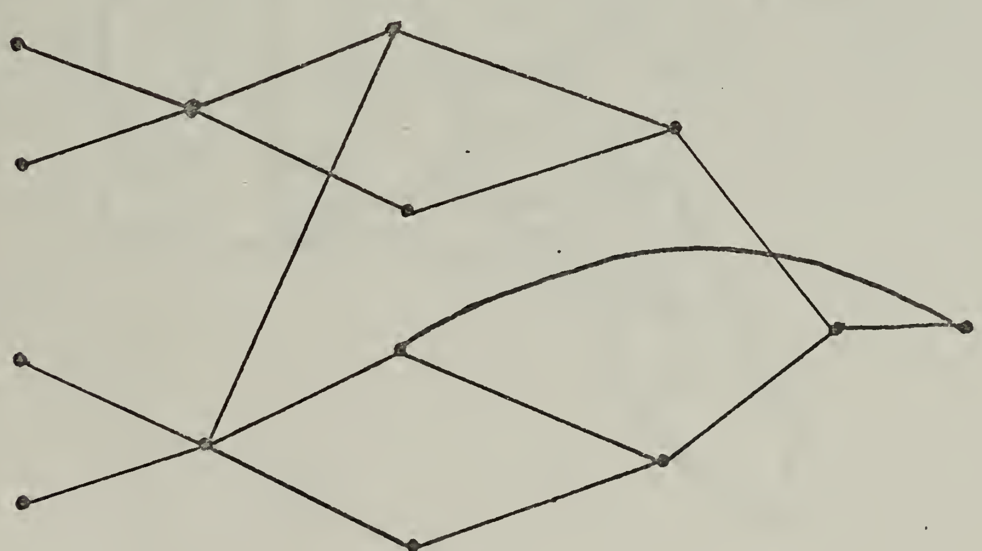

Figure 1. A Graph

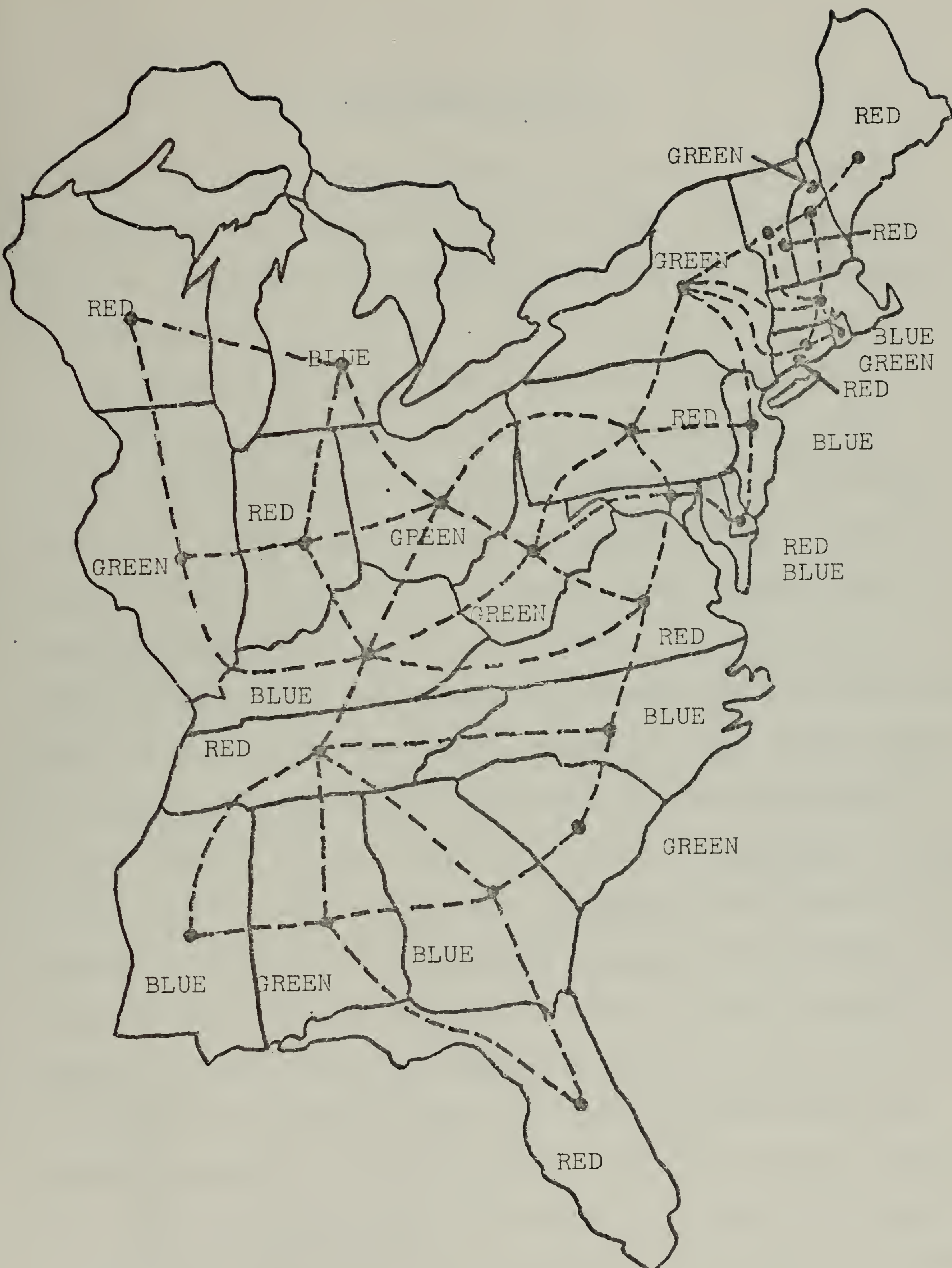

Figure 2. A Map and the Associated Graph

II. FUNDAMENTAL CONCEPTS

The definitions which follow are basic to graph theory and comprise the vocabulary for the graph coloring problem. They are essentially the definitions as stated by Busacker and Saaty [Ref.2].

Let V and E be sets. Let s and t be elements of V. The elements s and t may be associated together by the symbol (s&t), called the <u>unordered pair s and t</u>. The unordered pair s and t is equal to the unordered pair t and s (that is (s&t) = (t&s)). The symbol (V&V) denotes the <u>unordered product</u> of set V with itself. By definition (V&V) is the set of all possible unordered pairs of the elements of the set V. Let Γ be a mapping of set E into (V&V). An <u>undirected graph</u> is the mathematical system represented by the symbol (V,E,Γ). Set V is the set of <u>vertices</u> or <u>nodes</u> of the graph. Set E is the set of <u>edges</u> of the graph. The mapping Γ is called the <u>incidence mapping</u> of the graph. In some contexts there is no need to refer to the incidence mapping Γ explicitly. A graph will usually be denoted by G = (V,E,Γ), or by G = (V,E), when the incidence mapping remains implicit. If Γ(e) = (s&t) then the vertices s and t are called the <u>end points</u> of the edge e, s and t are said to be <u>adjacent</u>, and s and t are <u>incident</u> to the edge e. If the set of edges of graph G is empty, then, and only then, the graph G is said to be <u>degenerate</u>. If the sets V and E are

both finite then the graph G is called a finite graph. Otherwise it is said to be infinite. A graph G is complete if any two nodes are adjacent.

A geometric graph in Euclidean n-space is a set $V = \{v_i\}$ of points in Euclidean n-space and a set $E = \{e_j\}$ of simple curves satisfying the following conditions:

1. Every closed curve in E contains precisely one point of V.
2. Every open curve in E contains precisely two points of V and these agree with its end points.
3. The curves in E have no common points except for points of V.

Figure 3 depicts a geometric graph in Euclidean 3-space and illustrates the manner in which geometric graphs will be represented. By condition 3 above, the edges do not intersect except at end points. The interpretation of the apparent intersection of edges e_5 and e_3 is that they pass skew to each other in the third dimension.

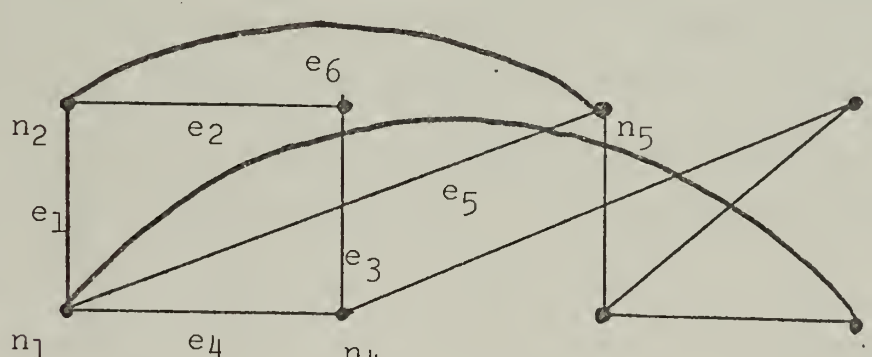

Figure 3. A Geometric Graph

A graph is planar if it can be represented in Euclidean 2-space. The graph of Figure 3 is planar as demonstrated in Figure 4.

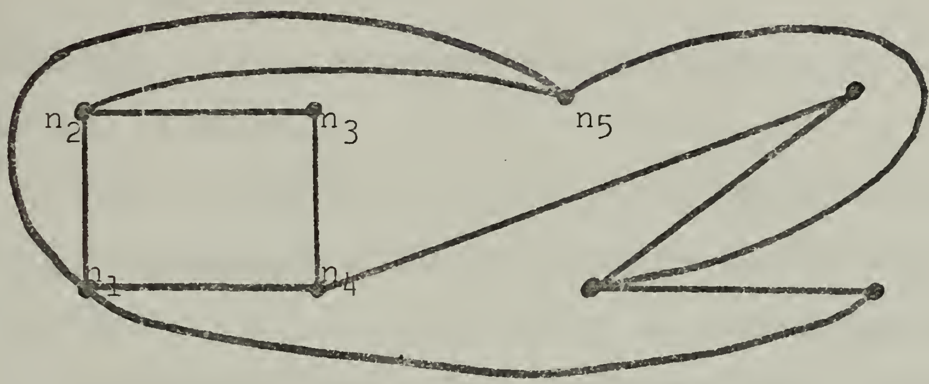

Figure 4. A Planar Graph

An important result of graph theory is that every finite graph G = (V,E) has a geometric realization in Euclidean 3-space. The proof is by construction. Select an arbitrary straight line in 3-space and call it L. Then corresponding to each vertex v of V select a distinct point of L. For each edge (v&w) of E select a distinct half plane in 3-space with L the boundary, and construct a simple curve on this half plane joining the points of L which correspond to v & w. It is clear that this construction satisfies the conditions for a geometric graph. As a result of this all finite graphs can be represented as in Figure 3.

Let $G = (V,E,\Gamma)$ and $G' = (V',E',\Gamma')$ be graphs. G' is a subgraph of G if and only if the following conditions are satisfied:

1. V' is a subset of V and E' is a subset of E.

2. If e is an edge of E' then $\Gamma(e) = \Gamma'(e)$.

3. If e is an edge of E' and $\Gamma(e) = (v\&w)$, then v and w are nodes of V'.

The nodes n_1 to n_5 complete with the edges e_1 to e_6 form a subgraph of Figure 3.

A finite sequence $e_1, e_2, \ldots, e_n$ of edges of a graph is an <u>edge progression</u> if there exists an appropriate sequence of vertices v_0 $v_1, \ldots, v_n$ such that e_i corresponds to $(v_{i-1}\&v_i)$ for $i = 1,2,\ldots,n$. The edges are not necessarily distinct and the vertices also are not necessarily distinct. The edge progression is said to be <u>closed</u> if $v_0 = v_n$ and <u>open</u> otherwise. A closed edge progression having no repeated edges is called a <u>circuit progression</u>. A <u>circuit</u> is any set of edges which, if properly ordered, form a circuit progression. In a geometric graph a circuit forms a closed simple curve. An open edge progression having no repeated edges is called a <u>chain progression</u>. A <u>chain</u> is a set of edges which, if properly ordered, form a chain progression. In a geometric graph, a chain is a set of edges which form an open simple curve. A <u>connected graph</u> is a graph such that every pair of distinct vertices are joined by at least one chain. In a geometric graph, there is at least one open curve between any two nodes of the graph. A <u>tree</u> is a connected graph which has no circuits.

A <u>dominating set of vertices</u> is a set W of vertices such that every vertex not in W is adjacent to a vertex in W.

A set of vertices such that no two vertices in the set are adjacent is called an <u>independent set of vertices</u>. A collection of sets is called a <u>partition</u> of the vertices of the graph if the union of all the independent sets equals the set of vertices and the intersection of any two of the independent sets is empty. A <u>coloring</u> of a finite graph is a partition of the vertices of the graph into independent sets. A coloring is demonstrated by coloring the nodes of the graph with colors such that if two nodes are in the same independent set then they will be assigned the same color.

Let n be the number of independent sets in some coloring of a graph. If n is less than or equal to the number of independent sets in any coloring of the graph then n is the <u>chromatic number</u> of the graph. In other words, the chromatic number of a graph is the minimum number of colors that are necessary to color the graph.

For certain classes of graphs, the chromatic number and an associated partition are known. In particular, the chromatic number of trees is 2. This will be demonstrated later. The 4-color problem is the conjecture that the chromatic number of planar graphs is 4. Figure 5 represents a planar graph with chromatic number 4 so that it is obvious that 4 colors are necessary. However, no planar graph is known for which 4 colors are not sufficient. Notice that specific planar graphs may have a chromatic number less than 4.

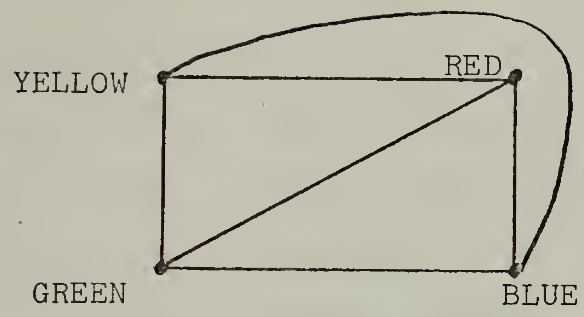

Figure 5. A Planar Graph with Chromatic Number 4

For more general graphs there are some basic results concerning the chromatic number. The first of these is Konig's Theorem which specifies that a graph is bi-chromatic if and only if it contains no circuits of uneven length. A sketch of the proof follows:

1. Color an arbitrary vertex y blue.
2. If a vertex x is colored blue then color all vertices adjacent to x red; and if x is colored red then color all adjacent vertices blue.

If some node x is colored both red and blue, then node x and the arbitrary node y are on a circuit of uneven length. If the graph is bi-chromatic, and it contains a circuit of odd length, then coloring the nodes of this circuit with alternating colors forces coloring one of these nodes with both colors.

Konig's Theorem is sufficient to prove that trees are bi-chromatic. Since trees have no circuits, they have no circuits of uneven length.

There are some known bounds of the chromatic number of arbitrary graphs. The chromatic number is certainly less than or equal to the number of nodes, since a trivial partition could be S(i) = {n(i)}, where n(i) is the i'th node.

A somewhat better upper bound is one more than the degree d of the node of largest degree. This is intuitively clear since no node is adjacent to more than d other nodes.

A yet superior upper bound is available. The bound is computed by the following procedure:

1. Order the nodes by their degree such that
   $d(i) \geq d(i+1)$.
2. Compute (for all i) the value of
   $S(i) = MIN(i, d(i)+1)$.
3. The upper bound is the maximum S(i).

This bound is not intuitively obvious. It may be thought of as a "kind of intersection of two sequences," in which the sequences are these:

1. The sequence of the integers from 1 to n, where n is the number of nodes of the graph.
2. The sequence of the degrees of the nodes plus one from the largest degree to the smallest.

For the nodes of largest degree (which are indexed lowest) the value MIN(i,d(i)+1) = i. For the nodes of smallest degree the value of the MIN function is d(i)+1. For a node indexed approximately n/2 and the succeeding node, the MIN function will step from the value of the index of the node

to the degree of the node plus one; and there is an upper bound on the chromatic number given at this "intersection."

An obvious lower bound to the chromatic number is 1 if there are no edges and 2 otherwise. Konig's theorem implies a lower bound of 3 if there is a circuit of uneven length.

The chromatic number is at least as large as the number of nodes in the largest complete subgraph of the graph. Clearly, a complete graph of n nodes requires n colors since any two nodes are adjacent. Moreover, by Brooks's Theorem [Ref.3], the chromatic number of a graph can not be less than the chromatic number of any subgraph.

Given any two positive integers d and c with d greater than c: then House [Ref.4] has constructed a graph, such that the chromatic number of the graph is d and the size of the largest complete subgraph is c.

One of the most straightforward methods known to color a graph is the Welch-Powell method.

The nodes of the graph are reindexed by the degree of the nodes. (The node of largest degree is assigned the lowest index.) Assign to Color(1) the first node. For all nodes v from index 2 to the last node, assign the node to Color(1) if it is not adjacent to any node already assigned Color(1).

Let set A be the set of all uncolored nodes. Let n be the first node of A. Assign n to the next available color. For all the remaining nodes in A, assign the node to this color if it has not already been colored, and if it is not

adjacent to any node assigned to this color. Let set A be the set of all remaining uncolored nodes, and continue this process until set A is empty.

Notice that the decision of which color to assign to a node is uncomplicated in this algorithm. Specifically, the node inherits the first color such that the node is not adjacent to some other node which already has that color.

This algorithm is very fast and usually produces good results.

III. ALGORITHM TO OBTAIN A COLORING

The proposed algorithm is presented in this chapter in two parts. Part 1 is a general overview of the entire procedure with the intent of demonstrating the logic flow and purpose of the steps of the procedure. Part 2 is a detailed description of the algorithm. It is listed in a step by step manner and the reader could easily draw up a flowchart of the method from this description.

The most basic goal of the algorithm is to determine a lower bound on the chromatic number by detecting a complete subgraph of the given graph. The nodes which are vertices of this subgraph are assigned distinct colors during this computation, and corresponding to each color, node sets are created. The result of this is displayed in Figure 6. The values of the colors are 1,2,3,...,k, where k is the size of

a complete subgraph of the given graph. Hence k represents a lower bound on the chromatic number. Associated with each color i are three sets, COLOR(i), LISTL(i), and LISTR(i). COLOR(i) is the set of nodes assigned the i'th color. Initially the only node assigned to COLOR(i) is the i'th node determined to be in the computed complete subgraph. LISTL(i) is a set of nodes; each node has the property that it is adjacent to at least one node in each COLOR(j) for all colors j which are less than i. LISTR(i) is a set of nodes such that each node has the property that it is adjacent to at least one node in each COLOR(j) for all colors j which are greater than i (and less than or equal to k). [Figure 6]. Node 5 is assigned to color 1, node 10 is assigned to color 2, and node 20 is assigned to color k. Node 2 is assigned to LISTL(2). This implies that node 2 is adjacent to node 5. Node 3 is adjacent to nodes 5 and 10. Node 5 is adjacent to 10, 15, 20, and all the nodes assigned colors from 2 to k-1. Node 8 is assigned to LISTR(1). This implies that node 8 is adjacent to nodes 10,15,...,20. Node 7 is adjacent to nodes 15,...,20. Notice that nodes assigned to LISTL(i) and LISTR(i) are not adjacent to the nodes assigned to COLOR(i). The nodes assigned to the sets COLOR(i) initially form the complete subgraph which is computed. Assuming that no changes are made to already colored nodes, observe that the nodes in LISTL(i) can not be given color values less than i. Similarly, the nodes in LISTR(i) can not be given color values greater than i.

	1	2	3	. . .	k
COLOR	5	10	15		20
LISTL	12 8	2	3		5
LISTR	8	7	12 3		4

Figure 6. Basic Tableau of the Proposed Algorithm

On the assumption that a coloring of k colors is possible, one concludes that node 8 must be assigned color 1, and node 3 must be assigned color 3. Therefore Node 8 is removed from LISTL(1) and is assigned to COLOR(1). Node 3 is removed from LISTL(3) and assigned to COLOR(3).

For the purpose of demonstration, assume that nodes 12 and 3 are adjacent. Then after node 3 is assigned to COLOR(3), the membership of node 12 in LISTR(3) is no longer valid, since a node belongs to LISTR(i) if it is not adjacent to any node in COLOR(i) but is adjacent to at least one node in each COLOR(j) for j = i+1 to k. Therefore node 12 is removed from LISTR(3) and placed in another LISTR(j) where j is less than 3. Thus, if node 12 is not adjacent to node 10, it would be placed in LISTR(2). If it is adjacent to node 10, then it would be tested for LISTR(1).

Assume that node 12 is adjacent to node 10 and is therefore placed in LISTR(1). Then since node 12 is in both

LISTL(1) and LISTR(1) it must be assigned to COLOR(1). This is accomplished and the process continues.

If at some point two nodes must both be assigned to COLOR(i), and the two nodes are adjacent, then it is not possible to color these nodes subject to the restriction of k colors. Consequently, an additional color is allocated, and one of the two nodes assigned to it. In Figure 7, nodes 4 and 5 must both be assigned to COLOR(3), and node 4 is adjacent to node 5. Hence, a new color (k+1) is allocated, and node 5 assigned to it.

In such a case, it is possible to recompute the upper bound (LISTR) for the uncolored nodes. Thus, node 6 which is assigned to LISTR(2) is not adjacent to node 5, and therefore belongs to LISTR(k+1). Therefore, following the allocation of a new color, the sets (LISTR) are recomputed for all the uncolored nodes. Notice that there is no effect on the lower bound (LISTL) of the nodes.

	1	2	3	. . .	k
COLOR					
LISTL			4 5		
LISTR			5 4		

Figure 7. Tableau Requiring a New Color

It may occur that at a given step there is no node which is forced to be colored due to membership in both LISTL(i) and LISTR(i). In this event a node is arbitrarily colored and the process of adjusting the sets(LISTL and LISTR) continues. The node which is arbitrarily colored is the first node of the first non-empty LISTL.

Since each time a node is assigned a color i it is deleted from the set LISTL(i), a sufficient condition for completion of the coloring process is that LISTL(i) be empty for i = 1,2,3,...,m (where m is the number of colors used).

In the more comprehensive description below, the general purposes of the 8 steps are:

Step 1. Initialize the algorithm.

Step 2. Compute a lower bound on the chromatic number, and form the sets COLOR(i) and LISTL(i) for each color of this lower bound.

Step 3. Form the sets LISTR(i) for each color.

Step 4. Color the nodes which are limited to only one possible color.

Step 5. Test for the need of allocating a new color, and if needed, perform the functions of allocating it, assigning the first node to it, and adjusting the sets LISTR.

Step 6. Perform the necessary arbitrary assignments of colors.

Step 7. Reassign nodes of LISTR(i) to other LISTR elements if the nodes are adjacent to the latest node assigned to COLOR(i).

Step 8. Reassign nodes of LISTL(i) to other LISTL elements if the nodes are adjacent to the latest node assigned to COLOR(i).

These steps require house-keeping operations in order to maintain control of the process. Therefore in the following paragraphs a detailed step by step description of the algorithm is given.

Step 1. Order the nodes of the graph such that node n precedes node m if the degree of node n is greater than or equal to the degree of node m. This ordering is a reindexing of the nodes.

Step 2. Compute a lower bound on the chromatic number of the graph. This lower bound is equal to the size of some complete subgraph of the graph. This is done as follows:

Step 2a. Set k = 1. The integer k will soon represent the computed lower bound, and will eventually be the number of colors used in partitioning the vertices.

Step 2b. Let V be the set of nodes of the graph. Since the nodes in V have been well-ordered by the index operation of Step 1, a smallest mode v_1 exists (in fact this is the node of largest

degree). Select this node and assign it COLOR(1).

Step 2c. Let $V' = V - \{v_1\}$.

Step 2d. Form set LISTL(1). This set is defined to be the set of all nodes w from V′ such that node v_1 is not adjacent to w.

Step 2e. Set A = V′ - LISTL(1).

Step 2f. If A = Ø then the graph is degenerate since the node of largest degree has degree 0. In the event the graph is not degenerate then the chromatic number is at least 2. Any node in set A along with node v_1 form a complete subgraph of size 2. Consequently, the value of k will be incremented to 2, and some node of A will be assigned COLOR(2). This is done in the following steps.

Step 2g. Increment k by 1.

Step 2h. Select the node with the smallest index n of set A and assign it COLOR(k).

Step 2i. A = A - {n}.

Step 2j. Form set LISTL(k). This set is defined to be the set of all nodes w from A such that node n is not adjacent to w.

Step 2k. Set A = A - LISTL(k).

Step 2l. Is set A = Ø? If not then any node in set A is adjacent to all the nodes which have been assigned colors and therefore along with those

nodes forms a complete subgraph. Therefore go to Step 2g to continue the computation of the lower bound. If set A is empty, then all the nodes of the graph have either been colored, or assigned to some LISTL(i). Considering the nodes which have been assigned to LISTL(i), it is clear by the construction that the minimum value of the color which may be assigned to them is i. This follows because any node in LISTL(i) conflicts with nodes assigned to COLOR(i) through COLOR(i-1).

Step 3. Having found a maximum bound on the chromatic number, and minimum bounds of the possible colors of all heretofore uncolored nodes, the next step is to compute upper bounds on all the uncolored nodes under the assumption that the minimum bound is in fact the chromatic number.

Step 3a. Let set A = the set of all uncolored nodes.

Step 3b. Let kp = k. The integer kp will represent a counter which will range from k to 1.

Step 3c. Set LISTR(kp) equal to the set of all nodes of A which are not adjacent to the nodes of COLOR(kp). That is, if node w is in A, and for all nodes n in COLOR(kp), w and n are not incident, then assign w to LISTR(kp). In addition, assign the value kp-1 to array element DOWN(w).

The purpose of this is for adjustment of the set LISTR at a later stage of the method.

Step 3d. A = A - LISTR(kp). The set COLOR(kp) has the property that for any node w in A, there exists some node n in COLOR(kp) such that w and n are adjacent.

Step 3e. kp = kp - 1.

Step 3f. Is kp = 0? If kp $\neq$ 0, then there remain sets LISTR(i), where i is between 1 and kp which are yet to be initially formed. Go to Step 3c. If kp is zero then all nodes which are not colored have been assigned to LISTR(i) for some i. The value of i represents an upper bound on the possible color of the node.

Step 4. Still operating under the assumption that the lower bound which has been computed is in fact the chromatic number, the algorithm continues by locating nodes which are not colored, but which have lower bounds on the possible color equal to the upper bounds on the possible color. Clearly, in this case, the necessary color to assign to these nodes has been determined.

Step 4a. Set kp = 1. kp will be a counter which will vary from 1 to k.

Step 4b. Set A = the intersection of LISTL(kp) with LISTR(kp).

Step 4c. If A = Ø then no node must necessarily be assigned color kp. Go to Step 4d which will increment kp and search for nodes which must be colored at the next higher color. If A ≠ Ø, then at least one node is both lower and upper limited by this color. Go to Step 4g, which will color the node, and accomplish some accounting.

Step 4d. Is kp = k? If so, then all the colors have been searched for some node to color, and none has been found. The next step will be to arbitrarily color a node, if more nodes remain to be colored. Go to Step 6. If kp ≠ k, then go to step 4e.

Step 4e. kp = kp + 1.

Step 4f. Go to Step 4b.

Step 4g. Select the node with the smallest index of the set A. (Call this node n.)

Step 4h. Assign n to COLOR(kl).

Step 4i. Since n has been colored, drop it from the set LISTL(kp). (That is LISTL(kp) = LISTL(kp) - {n}).

Step 4j. Since a new node has been assigned to COLOR(kp), it is very possible that there exists some node in LISTL(kp) which is adjacent to n. Consequently steps must be taken to ensure that LISTL(kp) is independent of COLOR(kp). Let set B be the

set of all nodes in LISTL(kp) which are adjacent to node n.

Step 4k. Is set B empty? If so then LISTL(kp) is independent of COLOR(kp). Go to Step 7. If B is not empty then the lower bound on the possible color of all the nodes in B will have to be increased. Go to Step 5.

Step 5. This step considers all the nodes in set B and deletes them from LISTL(kp), assigning them to a higher LISTL or possibly assigning them a color.

Step 5a. Select the node with the smallest index of set B. Call it w.

Step 5b. Is w an element of LISTR(kp)? If so, then the situation arises in which both node n and node w must be assigned to COLOR(kp) and they are adjacent. This means that a new color must be allocated, and one of these two nodes assigned to it. Go to Step 5c. On the other hand, if w is not an element of LISTR(kp), then it is only necessary to adjust the lower bound for the color of node w. Go to Step 8.

Step 5c. k = k + 1.

Step 5d. Assign node w to COLOR(k). Delete node w from set B, and from the set LISTL(kp).

Step 5e. At this point, since a new color has been allocated, and the new color has the highest

numeric value, it is possible that the upper bounds of some of the uncolored nodes may be changed. Specifically, if node v is not colored, and node v is not adjacent to node w, then the upper bound on the possible color of v is k. Set mkr = 1.

Step 5e1. For all nodes s in LISTR(mkr), if s is not adjacent to node w, then do the following. Assign s to LISTR(k), and delete it from LISTR(mkr). Set DOWN(s) = mkr. DOWN is an array which specifies that if s will have to be removed from LISTR(k), then the next lowest LISTR into which node s may possibly be placed is mkr, since mkr demonstrates that s is adjacent to some node in each of the COLOR sets from mkr + 1 to k - 1.

Step 5e2. mkr = mkr + 1.

Step 5e3. Is mkr = k? If so, then all the LISTR have been adjusted for the introduction of this new color. Go to Step 4k. If not, then go to Step 5e1.

Step 6. In the event that there are no nodes which have the property (at this state of the coloring) that they must be colored, then some node must be selected and colored independently of the forcing nature of the algorithm. These nodes to be arbitrarily colored are selected by the

decision rule that the first node of the first non-empty LISTL is used.

Step 6a. Set index counter kp = 1.

Step 6b. Is LISTL(kp) = Ø? If it is not, then the node of greatest degree in LISTL(kp) will be assigned the color kp. Go to Step 6e. Otherwise scan the remaining sets LISTL for some uncolored node.

Step 6c. Is kp = k? If so, then there is no uncolored node, and this means a solution has been found. Go to Step 9. If kp is less than k, then there remain more sets LISTL to consider.

Step 6d. kp = kp + 1. Then go to Step 6b.

Step 6e. Select the first node of LISTL(kp) and call it v. Go to Step 4h to assign this node to COLOR(kp).

Step 7. The purpose of this step is to adjust the set LISTR(kp), following the assignment of node n to COLOR(kp). The method is to locate all the nodes in LISTR(kp) which are adjacent to node n and then relocate them to another set of LISTR.

Step 7a. Let the set B equal the set of all nodes in LISTR(kp) which are adjacent to node n.

Step 7b. Is set B = Ø? If it is, then there is no adjustment to be made so go to Step 4. Otherwise, for all the nodes w in set B, perform the following procedure.

Step 7c. Set index counter j = DOWN(w) = the next lowest possible color to which node w may be assigned. From previous work it is known that node w is adjacent to some node in each of the sets COLOR(i) for i from DOWN(w) + 1 to kp.

Step 7d. Is node w adjacent to node n for some n in COLOR(j)? If so then decrease j by 1 and continue this step. (That is j = j - 1: Go to Step 7d.)

Step 7e. Node w is not adjacent to any node in COLOR(j), and this is the qualification to belong to LISTR(j). Assign node w to LISTR(j), delete w from set B, and go to Step 7b.

Step 8. The purpose of this step is to adjust the sets LISTL. Node w is adjacent to some node in COLOR(kp), and hence node w must be relocated to some new set LISTL(i) where i is between kp + 1 and k.

Step 8a. Set index counter j = kp + 1.

Step 8b. Is node w adjacent to node n for some n in COLOR(j)? If so, then increment j by 1 and go to Step 8b.

Step 8c. Node w is not adjacent to any node in COLOR(j). This is the qualification to belong to LISTL(j). Assign node w to LISTL(j), delete

w from LISTL(kp), delete w from set B, and go to Step 4k.

Step 9. The end of the algorithm.

IV. TEST COMPARISONS BETWEEN THE PROPOSED ALGORITHM AND THE WELCH-POWELL ALGORITHM

Since this proposed method does not guarantee a solution with the chromatic number, its value must be judged in comparison to other methods. Since the Welch-Powell algorithm is the most well-known, it was used as a benchmark.

Test graphs were generated as follows: Two nodes were adjacent if a generated random number was less than a specified factor FR which represents a probability. Six sets of 100 graphs were generated. Table I summarizes the data about these graphs, and it presents the distribution of differences between the number of colors used by the Welch-Powell algorithm and the number of colors used by the proposed algorithm.

Three general conclusions can be reached from this data:

1. For an arbitrary random graph, the probability is approximately 65% that the new algorithm will produce a better result than the Welch-Powell algorithm.
2. The average difference in colors increases as the number of nodes increases.

3. The average difference in colors increases as the probability of conflict increases.

D.C. Wood [Ref.5] published an algorithm in 1969. Wood arranged his test data in the form of results which were better, equal, or worse than the Welch-Powell results. Table II presents the summary of Table I and Wood's results in a concise form.

A test was conducted of fifty graphs of twenty nodes. The probability of conflict was incremented from .30 to .80. The results are presented in Table III and tend to substantiate the conclusion that smaller graphs are approximately equally colored by the methods.

Finally, an additional fifty graphs were generated and time comparisons of the new method and the Welch-Powell method were made. The graphs ranged from 40 to 90 nodes with the probability of conflict from .20 to .70. These results are tabulated in Table IV. The times are measured in seconds and RATIO is the quotient of the time required by the new algorithm to the time required by the Welch-Powell algorithm. In general, the time required by the new algorithm increases directly as the probability of conflict and the number of nodes increases.

NODES	FR	-5	-4	-3	-2	-1	0	1	2	AVG
50	.25				8	41	43	7	1	-.48
	.50		1	1	18	45	30	5		-.83
	.75		1	3	25	36	26	8	1	-.89
100	.25			1	19	47	31	2		-.86
	.50		1	12	26	32	23	6		-1.18
	.75	2	9	14	23	24	16	11	1	-1.45

Table I.

NODES	FR	BETTER		SAME		WORSE	
		NEW	WOODS	NEW	WOODS	NEW	WOODS
50	.25	49	8	43	66	8	26
	.50	65	30	30	52	5	18
	.75	65	48	26	31	9	21
100	.25	67	7	31	20	2	73
	.50	71	13	23	50	6	37
	.75	72	72	16	20	12	8

Table II.

NODES	FR	W_P - NEW				
		-2	-1	0	1	2
20	.3-.39			9	1	
	.4-.49		3	7		
	.5-.59		2	7	1	
	.6-.69	1	3	6		
	.7-.79		3	7		

Table III.

# NODES	WELCH-POWELL COLOR	PROPOSED COLOR	W_P TIME	NEW TIME	RATIO
40	8	7	.24	.68	2.83
40	9	8	.35	1.01	2.89
40	10	9	.31	1.09	3.52
40	12	11	---	1.14	----
40	15	13	.38	1.29	3.39
50	8	7	.28	.92	3.28
50	10	10	.33	1.47	4.45
50	12	11	.41	1.27	3.10
50	14	14	.45	2.12	4.71
50	17	15	.48	2.03	4.23
60	9	9	.39	1.62	4.15
60	12	11	.45	1.88	4.18
60	14	14	.50	2.5	5.00
60	16	15	.55	3.02	5.49
60	19	18	.65	3.93	6.05
70	10	10	.47	2.02	4.30
70	13	11	.56	2.16	3.86
70	14	13	.61	2.71	4.44
70	17	19	.69	4.79	6.87
70	22	21	.81	5.88	7.26
80	11	11	.59	2.96	5.02
80	14	13	.67	3.02	4.51
80	17	15	.72	4.16	5.78
80	21	19	.91	5.83	6.41
80	23	24	.97	8.21	8.46
70	4	3	.38	.96	2.53
70	6	5	.46	.96	2.09
70	8	8	.48	1.85	3.85
70	12	11	.62	----	----
70	17	17	.79	4.83	6.11
75	4	3	.89	.97	2.99
75	6	6	.97	1.9	2.98
75	10	8	.61	2.0	3.28
75	11	11	.58	3.11	5.36
75	18	16	.94	4.66	6.96
80	4	3	.45	.91	2.02
80	1	7	.43	1.98	4.60
80	9	8	.87	2.16	2.48
80	11	12	.59	3.62	6.14
80	19	18	.93	7.14	7.68
85	4	4	.44	1.06	2.41
85	7	6	.95	1.59	1.67
85	9	9	.58	2.74	4.72
85	12	11	.80	3.51	4.39
85	21	19	1.15	7.79	6.77
90	9	9	.44	1.68	3.82
90	7	7	.55	2.13	3.87
90	10	9	.63	2.77	4.43
90	12	12	.72	4.06	5.64
90	21	21	1.09	9.20	8.44

Table IV.

V. THE SCHEDULING PROBLEM

Let C be a set of classes offered during any academic period at some institution. In anticipation of course demands, many of the classes may be the same course. A *segment* is a class which is distinguished from all the other classes of the same course.

Each class in C requires time during the period of one week. An *n-hour class* will require n 1-hour periods per week. Furthermore, it will be assumed that no two of the periods will be on the same day. An *h-lab class* will require h 1-hour periods to be consecutive on the same day unless h is larger than 3, in which case the h-lab class will require (h/2) consecutive periods on two days. For example, a 3-hour class requires three 1-hour periods per week with each period on different days. A 3-lab class requires one day with three consecutive 1-hour periods. A 4-lab class requires two days with each day to have two consecutive 1-hour periods.

The time requirements for each class can be specified as a pair (n,h). A class which is both an n-hour class and a h-lab class may be treated as two classes - an n-hour class and a h-lab class.

Let S be the set of students and instructors. Associated with each person s in S is a subset of C, $C_s = \{c: \ s \text{ is in class } c\}$. That is, each person will

engage in some of the classes. It is assumed that all the instructors and students have been properly assigned to specific segments of the classes.

Let c(1),c(2),...,c(n) be the set of segments associated with any person. Then it is clear that no two of these segments may convene simultaneously. The Scheduling Problem is the problem of assigning time periods to the segments of C such that:

1. All class segments must be assigned to time periods which occur within a specified limits. For the test data the maximum number of available days per week was 5 and each day was limited to 9 periods.
2. The time requirements for the classes are satisfied.
3. If s is a person with class segments c(1) and c(2), then c(1) and c(2) can not be scheduled at the same time.

The scheduling problem can be further stipulated to satisfy additional constraints or requirements. Examples of some of the possible additional requirements may be:

1. That the hour assigned to each segment remains the same for each day. That is, a 3-hour class assigned to Monday, Wednesday, and Friday must convene at the same time each of these days.
2. No person may attend more than three consecutive hours of class in any day.
3. At least one half-day period be unassigned for each person.

The Conflict Graph of the schedule is a graph whose nodes are the segments and whose edges are computed as follows: For any person s and for any pair of segments c(1) and c(2) which are taken or taught by s, c(1) and c(2) are adjacent. Thus the conflict graph is the graph whose edges represent conflicts between the segments.

The problem of assigning periods to the segments is seen to be equivalent to the problem of assigning colors to the nodes of the conflict graph. The scheduling problem is solvable by assigning periods to the classes and distributing the periods over the days of the week and the hours of the day in accordance with the requirements of the classes. Thus, the colors of the conflict will be translated into two-component vectors of the form (day, time). Additionally, as many vectors as the requirements of the classes dictate, will be assigned. For an example, a 3-hour class will receive three colors identified as (day(1),time(1)), (day(2),time(2)), and (day(3),time(3)), where day(1), day(2), and day(3) are pairwise unequal.

A solution was attempted to the scheduling problem with the additional requirement that each class convene at the same hour each day. In the above example, time(1) = time(2) = time(3).

The method used is illustrated in Figure 8. The first step is to schedule the lab classes by coloring them by the unmodified coloring procedure. Next, these colors were

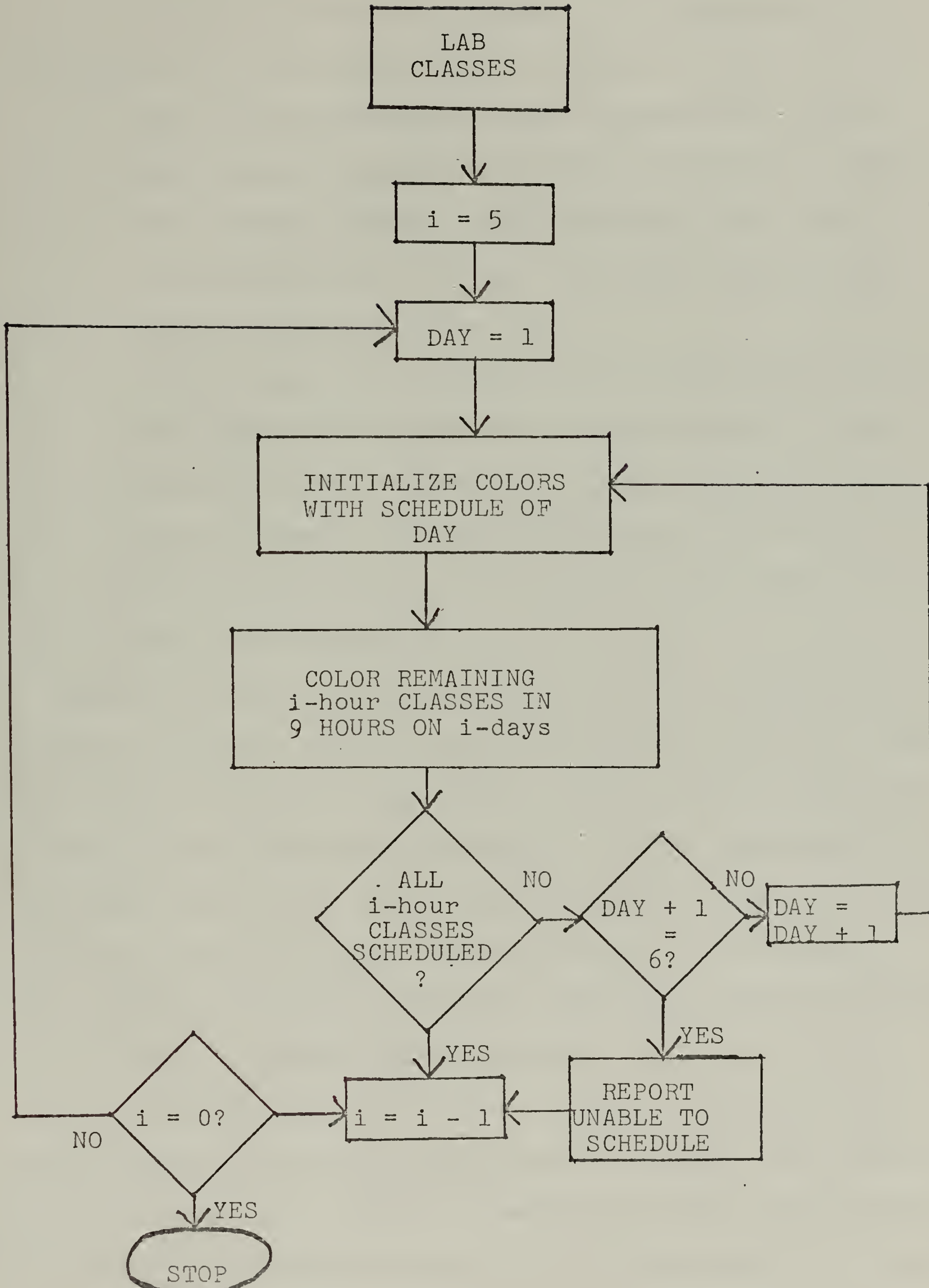

Figure 8. Basic Flowchart of Scheduling Problem Solution

transformed into appropriate pairs which represent the weekly schedule of these lab classes. The purpose of scheduling the lab classes in this manner is that:

1. Due to the consecutive time requirements, a lab must satisfy the most demanding conditions. Rather than making numerous modifications to the basic coloring algorithm it was felt that special treatment be awarded to the labs.
2. In the event that the additional requirement of at least one half-day period be unscheduled for each student, then that period could be considered as a lab of 5-hours consecutive duration. Thus this time could be controlled prior to application of the coloring method.

Next the 5-hour classes are scheduled. These classes present the most demanding hour requirements after the labs. The reason is that each of these classes must be scheduled on each of the five available days. If the algorithm for coloring the conflict graph were to operate on all the classes at once, then the possibility of two and three hour classes pre-empting the time the 5-hour classes require, would render a schedule unacceptable.

In order to color the 5-hour classes, the algorithm is supplied with an initial assignment of colors. This initial assignment of colors is the schedule which has already been determined for the day of the week which is identified by DAY and is the schedule of lab classes for DAY = 1 (in this

case). The 5-hour classes must be scheduled on this day subject to the schedule which has already been determined.

The coloring algorithm is further modified to verify that a class can be scheduled on the required number of days before it is colored. The class is then assigned to the correct number of days of the week at the time which is directly represented by the color. In the event that a sufficient number of days at that time is not available, the algorithm marks the time as not possible, and adjusts the possible times LISTL and LISTR in the algorithm). In essence then, the algorithm is modified to operate in the following way: The algorithm accepts an initial coloring which is the schedule of the weekday labeled DAY. The method then proceeds normally except that prior to any class getting scheduled, a scan is made of all days to verify the required number of hours for the class. If the hour requirement can not be satisfied, then the algorithm treats the situation as a conflict at that hour and the assignment is not made. If the hour requirement can be satisfied, the class is assigned to DAY and the requisite additional days.

After the 5-hour classes are scheduled, the 4-hour classes are scheduled. Again the algorithm is supplied with an initial coloring which represents the schedule of the first day. This schedule includes the labs and the 5-hour classes. The algorithm operates in the same manner as described for the 5-hour classes. However if some

classes remain unscheduled, the value of DAY is incremented by one and then represents the second day of the week. Some 4-hour classes may not have been able to be scheduled on Monday but can be scheduled on Tuesday. The algorithm is supplied with the schedule of the second day and the remaining 4-hour classes are scheduled. The advantage of this is that at some particular hour of day one, the only conflict with a particular 4-hour class may be a lab class. Then for the subsequent days, that is, days two through five, there is no conflict, and this second pass through the algorithm will schedule the class at that time.

Following the 4-hour classes, the 3-hour classes are treated in the same manner. Also, those 3-hour classes unscheduled on day one are scheduled on day two, and in the event that classes remain unscheduled, they are scheduled on day three. Notice that days three, four, and five are still sufficient to schedule the 3-hour classes.

This same process continues until all the classes have been scheduled.

In the unfortunate event that a class can not be placed in the schedule due to conflicts with previously scheduled classes, it is reported as unscheduled.

The algorithm was developed and tested on the class data of two of the quarters of the Naval Postgraduate School. Unfortunately, the method failed to produce a valid schedule for either of the quarters. In the first case,

with 424 classes to be scheduled, the algorithm left from 3 to 7 classes unscheduled. In the second case, with over 500 classes to be scheduled, the algorithm left from 13 to 21 classes unscheduled.

However, the results are not totally poor. In the first case a hand modification produced an acceptable schedule. This hand modification violated the requirement that classes convene at the same hour each day. It is felt that this violation is acceptable for a small number of classes.

The conflict graph coloring method was utilized to produce the schedule of finals for one of the quarters for the Naval Postgraduate School. The final schedule problem is not as demanding as the quarterly schedule in that each class only requires one period. Thus, the final schedule can be produced by the unmodified coloring algorithm. There was one additional constraint on the finals schedule. This constraint was that no person be scheduled for more than two finals in any day. A simple manual procedure was used to verify this condition.

BIBLIOGRAPHY

1. Welch, D.J.A. and Powell, M.B., "An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems," Computer J., Vol. 10, No. 1, pp. 85-86, May 1967.

2. Busacker, R.G., and Saaty, T.C., Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill Book Co., Inc., New York, 1965.

3. Brooks, R.L., "On Colouring the Nodes of a Network," Proc. Cambridge Philos. Soc., Vol. 37, pp. 194-197, 1941.

4. House, L.C., "A k-Critical Graph of Given Density," American Mathematical Monthly, Vol. 74, No. 7, Aug-Sep 1967.

5. Wood, D.C., "A Technique for Colouring a Graph Applicable to Large Scale Timetabling Problem," British Computer Journal, Vol. 12, No. 4, Nov 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assoc Professor U.R. Kodres, Code 53 Kr Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LCDR E. Singer, USN ASW Systems Project Office Manager Advanced Command & Control Systems Washington, D.C. 20360	1
5.	LT Robert A. Draper, USN 406 Kenmore Avenue Sunnyvale, California 94086	1
6.	Dean Brooks J. Lockhart Code 022 Naval Postgraduate School Monterey, California 93940	1
7.	Michael G. Corcoran Code 0301 Naval Postgraduate School Monterey, California 93940	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A Graph Coloring Algorithm and A Scheduling Problem

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; June 1971

5. AUTHOR(S) (First name, middle initial, last name)

Robert A. Draper

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1971	47	5

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

The graph coloring problem is defined, and its importance in several applications is noted. A new algorithm to color graphs is presented and tested against the Welch-Powell algorithm. Significantly better results are obtained on a sequence of three hundred randomly generated graphs.

The new algorithm is applied to the solution of a scheduling problem.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Graph						
Graph Coloring						
Scheduling						
Time Tabling						

Thesis 128126
D729 Draper
c.1 A graph coloring
algorithm and a sche-
duling problem.